

CS21: INTRODUCTION TO COMPUTER SCIENCE

Prof. Mathieson

Fall 2018

Swarthmore College

Outline Nov 16:

- Quiz 4: first 25-30min
- Runtime of bubble sort and selection sort
- Insertion Sort demo
- Runtimes in action
- Can we create a faster sorting algorithm?

Notes

- Lab 8 due **tomorrow** night!
- Office hours **today** 3-5pm, Ninja session tonight
- Lab 9 posted soon, due Mon after Thanksgiving

Sorting

3 sorts for this class

- **Selection Sort**: iteratively find the minimum element and place it in the correct position
- **Bubble Sort**: move through the list, swapping adjacent elements if they are out of place (repeat until sorted)
- **Insertion Sort (didn't get to)**: for each element of the list, move it down until it is in the correct position

Types of sorting

- **Out-of-place**: leaves the original list alone and creates a new sorted list (returns new list)
- **In-place**: modifies (mutates) the original list via swaps so that it is now sorted
- **Pros of in-place sort**: no new data structure needed (saving space)
- **Cons of in-place sort**: original order destroyed (in some cases it might be important), can be more difficult to implement

Runtime of our sorting algorithms

bubble sort

$j=0$ $j+1=1$

7 3 9 2 4

3 7 9 2 4

3 7 2 9 4

3 7 2 4 9

3 2 7 4 9

3 2 4 7 9

2 3 4 7 9

$i=n-1$
of comparisons

$i=n-2$

i	j
$n-1$	0 1 2 ... $n-2$
$n-2$	0 1 2 ... $n-3$
$n-3$	0 1 2 ... $n-4$
...	...
3	0 1 2
2	0 1
1	0

of comparisons

$n-1$
 $n-2$
 $n-3$
...
3
2
1

Sum

$n+n+n \dots n$

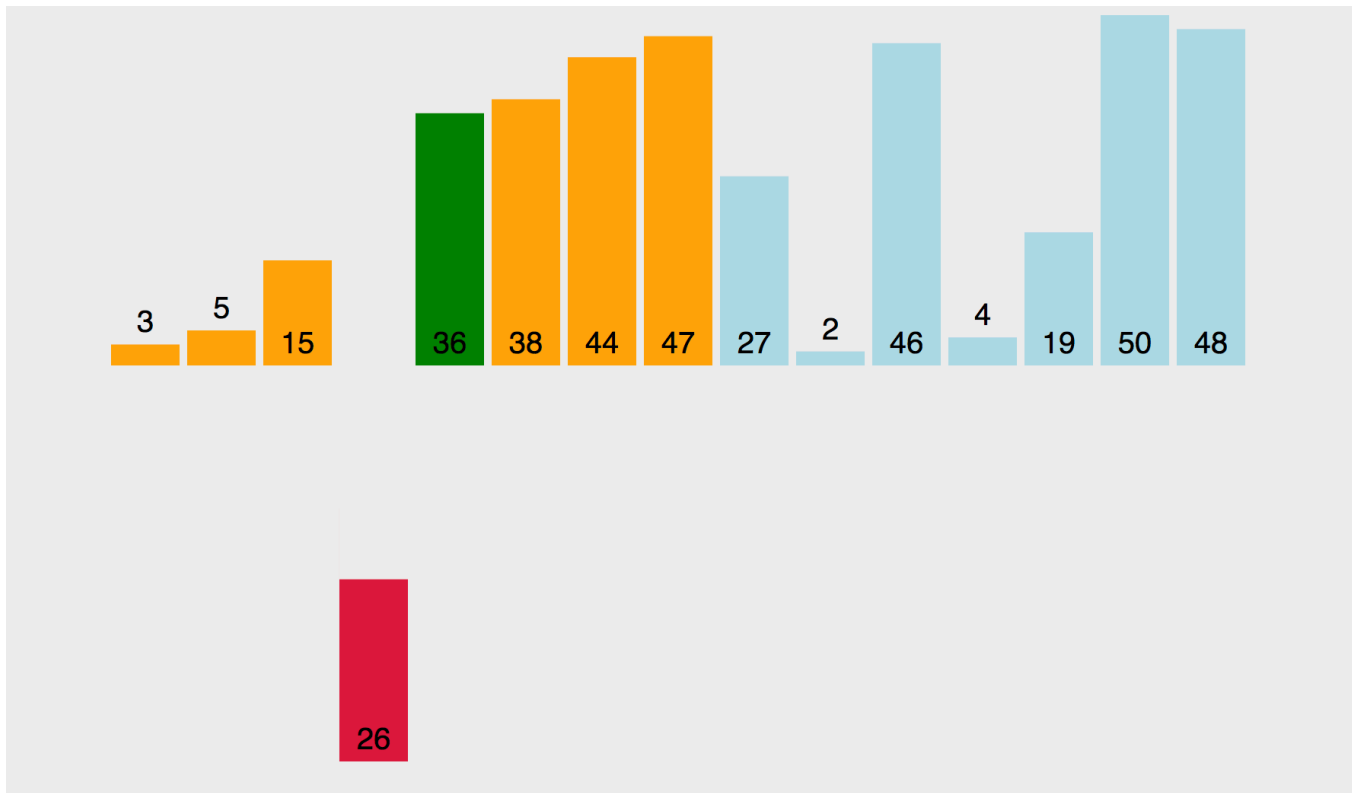
$\frac{n(n-1)}{2}$

order $\Rightarrow O(n^2)$




Insertion Sort demo

- <https://visualgo.net/bn/sorting>



Runtime Comparison demo

- <https://www.toptal.com/developers/sorting-algorithms/>

 Play All	 Insertion	 Selection	 Bubble	 Shell	 Merge	 Heap	 Quick	 Quick3
 Random								
 Nearly Sorted								
 Reversed								
 Few Unique								

Runtime in action

- `/cs21/inclass/w10/sort_runtime.py`
- Idea: if we double the length of the list, we should see the runtime quadruple (x4)

sort_runtime.py example output

```
Number of elements: 2000
Runtime of selection sort: 0.15 seconds

Number of elements: 4000
Runtime of selection sort: 0.55 seconds

Number of elements: 8000
Runtime of selection sort: 2.16 seconds

Number of elements: 16000
Runtime of selection sort: 8.81 seconds

Number of elements: 32000
Runtime of selection sort: 35.08 seconds
```

Can we do better than $O(n^2)$?

- Idea: thinking along the lines of binary search, what if we could divide the list in half and sort both pieces, then merge them together?